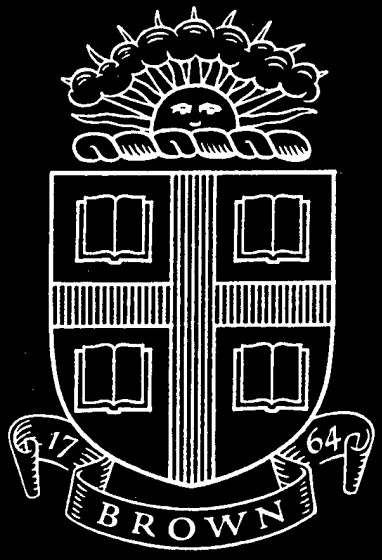
Codes for Optimal Stochastic Control: Documentation and Users Guide

Dennis Jarvis and Harold J. Kushner

May 1996 LCDS #96-3



Lefschetz Center for Dynamical Systems
and
Center for Control Sciences

Division of Applied Mathematics Brown University Providence RI 02912

Codes for Optimal Stochastic Control: Documentation and Users Guide

Dennis Jarvis and Harold J. Kushner

May 1996 LCDS #96-3

Codes for Optimal Stochastic Control: Documentation and Users Guide

Dennis Jarvis*and Harold J. Kushner,†
Division of Applied Mathematics
Brown University
Providence R.I., 02912

May 1996

*This work was partially supported by (ARO) DAA-H04-96-1-0075 and AFOSR-91-0375

†The work was partially supported by AFOSR-F49620-92-J-0081 and NSF ECS-9302137.

Abstract

This report documents codes for the numerical solution of control and optimal control problems for diffusion or reflected diffusion models in dimensions two to four and for continuous time Markov chain control problems where the state space of the chain is a grid in such a Euclidean space. The control appears linearly in the dynamics and cost function but otherwise the process and cost function are general. The underlying numerical methods use efficient forms of the approximation in policy space and multigrid type methods, based on the Markov chain approximation method of [7].

1 Introduction

Numerical methods of stochastic control have been successfully applied in many areas. The further use of these techniques requires the availability of easy to use and flexible codes, which can be adapted to a variety of basic problems. This report documents a set of codes which were developed originally for the numerical study of various problems in modern telecommunications. Those applications were quite successful [5, 8]. These references discussed a versatile numerical method for getting good approximations to queueing and multiplexing systems and getting optimal controls, and various simple approximations to them, as well as for experimenting with approximation schemes when the systems data are not well known. Much supporting numerical data was given and it was shown that the technique can be of great help in the design and analysis of systems.

The codes developed for those applications are quite general in nature and can be applied to a great variety of stochastic control problems with diffusion or reflected diffusion type models. They can also be used for the solution of control problems for continuous time Markov chain models whose state spaces are regular grids in Euclidean space.

This report is a documentation and user's guide to the codes for workstations. The user must, of course, supply the data for the system and cost function of interest. The format will be discussed in detail in the report and will be illustrated by concrete examples. On the boundary of the state space, the process can be either reflecting or absorbing, as desired. The program can be used in many ways, as described in the sequel. The choices are indicated to the program by an options parameter, which is entered at run time. The options include various choices over the input and output. In many problems the cost function of interest is the sum of several terms. While one might want the optimum policy and cost for the total cost, the individual components of the cost are often of great interest as well. These can be evaluated under the optimal control. Additionally, one has the choice of not optimizing, but merely evaluating the cost (and its components) under a given control. Threshold controls are of great interest, and it is easy to use the code to evaluate performance under such fixed controls. In addition, the numerical solutions and controls can be saved for postprocessing analysis such as control curve plotting. More details are given in Section 3.6 which lists the run time options. The flexibility offered by this method allows a

wide range of problems to be computed with hopefully minimal effort. High performance machines such as the Cray C90 require (limited) modifications to achieve effective vector performance and reduced execution times.

Efficient numerical solution techniques are used to get the various values of interest and the optimal controls. The original process and cost function are approximated by the versatile Markov chain approximation method. The basic numerical scheme then uses the approximation in policy space method for solving these approximating control problems. This approximation in policy space method generates a minimizing sequence of policies. For each such policy, there is a linear system of equations which must be solved, and this is the core of the computational burden. This system of equations represents the cost function for a Markov chain control problem. Experience has shown that overrelaxed Gauss-Seidel iterations combined with multigrid methods work very well, and (generally) significantly reduce overall computation time. Multigrid use without overrelaxation also works well. The software allows the user to tailor the computations by specifying the number of multigrid sublevels and the overrelaxation parameter for each level.

The basic diffusion model is discussed in Section 2. The description centers around the process with reflecting boundaries since it is more complicated than the absorbing boundary case. For the reflecting boundary case, the cost can be either of the ergodic or discounted type. The details concerning the structure of the state space and the basic numerical methods are similar to those for the absorbing boundary problem, and the few changes and simplifications (adding the absorbing boundary cost and dropping the reflection terms and associated costs) are mentioned at the end of the section. The state space is a hyperrectangle in all cases. Section 3 is concerned with the definitions needed by the program, and the input data formats and the files which the user provides. The codes are very flexible, and the structure was developed so that many types of problems could be accommodated. Section 3 also explains how to compile and run the program, and how to select the various options, as well as the possible outputs. A detailed example is provided to illustrate the procedure. Sections 4 and 5 contain other illustrative examples. The above examples are for the reflecting boundary case. An example for the absorbing boundary case is provided in Section 6. Section 7 deals with the continuous time Markov chain control problem. The procedure is essentially the same as for the diffusion model, except that the controlled transition rates must be defined. Section 8 contain an example of

a continuous Markov chain problem with controlled transition rates.

2 Process Model and Cost Function Descriptions

The codes can be used for the numerical solution of optimal control problems where the underlying model is of the diffusion or reflected diffusion type in dimensions two to four and for the calculation of costs or cost functions associated with given controls. They can also be used for the Markov chain models where the state space is a regular grid in a Euclidean space of dimensions two to four. In the next subsection, we describe the diffusion models. In all cases, the state space is a hyperrectangle G in Euclidean $d-$space, $d = 2, 3, 4$. The particular control problems which originally motivated the development of the codes arose in queueing or telecommunications situations, where the natural state space was often a hyperrectangle. In these applications, the states frequently correspond to (scaled) buffer or queue occupancies, and hence are non-negative. There are also physical upper bounds, which yield the upper bounds of the confining rectangle. The reflection directions on the boundary were determined by the physics of the flow within the network and were constant on each face of the hyperrectangle. The present code keeps the general structure, hence the state is confined to some hyperrectangle.

To do numerical work, one needs to work in a bounded state space. If the basic state variables in a model do not have natural finite bounds, then they generally need to be bounded in some fashion, so that "finite" procedures can be used. The most common approach is the artificial (upper and lower) truncation of each state at some appropriately large values. To prevent the process form exceeding these values, one uses either a reflection or absorption on the boundary. One tries to select the boundary behavior such that it does not seriously affect the numerical data of most importance. Thus, even for general numerical problems, one might still wish to work with a state space which is a hyperrectangle, with appropriate boundary conditions imposed.

The hyperrectangle state space is defined by

$$G = \{x : X_i^- \leq x_i \leq X_i^+\},$$

where X_i^+, X_i^- are real numbers. The cost function for the reflecting or absorbing boundary diffusion model is described below in the next subsection.

2.1 Diffusion Models: Reflecting Boundaries

Structure of the process model. The drift term in the diffusion can have an arbitrary dependence on the state x, but it is assumed to be linear in the control, and the covariance matrix is a constant. For the reflecting boundary case, the process is reflected "inwards" when it tries to leave G. More particularly, there are vectors p_i, and q_i, which are the reflection directions on the surfaces $x_i = X_i^-$ and $x_i = X_i^+$, resp. These are not necessarily unit vectors, and we will now explain how they are specified. Define the matrices $P = \{p_1, \ldots, p_d\}$, $Q = \{q_1, \ldots, q_d\}$, where the p_i, q_i are the columns. Refer to Figure 1 which illustrates a two dimensional problem. Suppose that we are sitting at a point x on a face of G and move out of G one "unit" in the direction orthogonal to the surface, and denote the new point by y. The p_i, q_i are the vectors needed to return us to the face (hyperplane) on which x lies when we are at y and move in the desired reflection direction. The reflection direction depends on the face on which x lies, but not otherwise on x, i.e., they are constant on each face. This is the typical setup in problems which arise in queueing theory. For one example, let $d = 3$, and suppose that the face is defined by $x_1 = X_1^+$. Then the direction vector q_i takes the form $q_i = (-1, q_{12}, q_{13})$, where $y + q_i$ is on the hyperplane defined by $x_1 = X_1^+$. Note that the first component must be negative. If the hyperplane is defined by $x_1 = X_1^-$, then we have $x = y + p_1$, where $p_1 = (1, p_{12}, p_{13})$. Note that the first component must be positive. With this format, we must always have $|q_{ij}| \leq 1$, $|p_{ij}| \leq 1$.

The general form of the diffusion process can be written as

$$dx = b(x, u(x))dt + \sigma d\bar{W} + PdL + QdU. \tag{2.1}$$

The term $b(x, u)$ is referred to as the "drift function." The drift is linear in u : $b(x, u) = b(x) + Ku$, where K is a matrix of real numbers, $b(\cdot)$ is continuous and $u(x)$ is a feedback control with M real valued components $u(x) = (u_1(x), \ldots, u_M(x))$, where $0 \leq M \leq 4$. There are $\bar{u}_i$ such that $0 \leq u_i(x) \leq \bar{u}_i$. The $\bar{W}(\cdot)$ is a standard Wiener process (covariance matrix being the identity). The covariance matrix $\Sigma = \sigma\sigma'$ is a constant. The L, U are vectors whose components $L_i(\cdot), U_i(\cdot), i = 1, \ldots, d$, are non decreasing real valued processes: $L(0) = U(0) = 0$, and $L_i(\cdot)$ can increase only at those

t where $x_i(t) = X_i^-$, and $U_i(\cdot)$ can increase only at those t where $x_i(t) = X_i^+$. It is always assumed that $b(x)$ and the direction vectors p_i, q_i satisfy what is required for the control problem to be well defined [7].

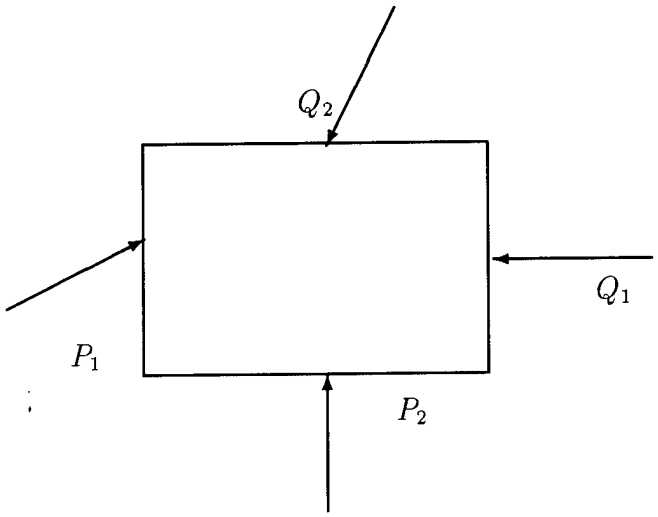


Figure 1. Reflection directions.

The cost function. For a continuous function $k(\cdot)$, define the "cost rate" in the form

$$k(x,u) = \sum_{i=1}^{N} \bar{c}_i k_i(x(t)) + \sum_{i=1}^{M} c_i u_i(x(t)), \quad M \leq 4. \tag{2.2}$$

The cost function can be of either the ergodic or the discounted forms. In program usage, the $\bar{c}_i, c_i$ are combined into the coefficients $c(1), c(2)$, etc. For real numbers l_i, v_i, define the ergodic cost function

$$\gamma(u) = \lim_T \frac{1}{T} E \int_0^T \left[k(x(t), u(x(t)))dt + \sum_i (l_i dL_i(t) + v_i dU_i(t)) \right] \tag{2.3}$$

and the discounted cost function

$$W(x,u) = E \int_0^\infty e^{-\beta t} \left[k(x(t), u(x(t)))dt + \sum_i (l_i dL_i(t) + v_i dU_i(t)) \right], \tag{2.4}$$

where the initial condition is $x(0) = x$.

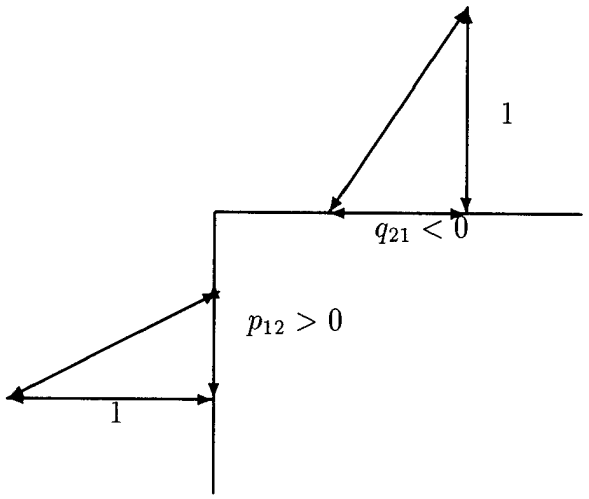


Figure 2. Components of Reflection Directions

Computing normalized "components" of optimal cost. In many optimization problems the cost criterion is the sum of several components. While the minimum value is important, the values of the components themselves might also be important. For example, in a queueing problem the total cost might be the sum of its components: one which measures delay and another which measures losses. It is worthwhile to know what the mean delay and the mean loss are, under the control which minimizes the sum of the mean values.

Suppose that the optimal control and cost have been calculated. One of the options available in the program is the computation of the values associated with the components of the costs: the values of (2.3) or (2.4) with the optimal control used but with the individual components of the cost replacing the bracketed quantity in (2.3) or (2.4). For specificity, let us restrict out attention to (2.3). If computation of the cost associated with the components is requested (see below for instructions on how to make this request of the program), the cost due to all the individual L_i, U_i in (2.3) with nonzero coefficients (l_i, v_i, resp.) will be computed (but with the coefficient l_i, v_i set equal to unity). Similarly, for the values of individual k_i and u_i components, for all i such that $c_i \neq 0$ or $\bar{c}_i \neq 0$. The total number of nonzero values of $c_i, \bar{c}_i$ can be at most $3 * dim$, where dim = dimensionality of the

model.

The program has been designed to be flexible and easy to use. In order to allow for reasonable generality without excessive code complexity, the program has been organized so that the user supplies $b(x,u)$ and $k(x,u)$. This is done in a simple Fortran code, as will be seen in the examples below. The program contains numerous options from which the user can choose to tailor the computation to a particular problem.

Comments on the numerical method. For the numerical solution, the diffusion model (2.1) with the given cost function is approximated via the Markov chain approximation method. See [6, 7] for discussion of the general method. The latter reference contains many applications, and details concerning approximations and numerical methods. At present, the Markov chain approximation method seems to be the method of choice for getting the optimal costs and controls as well as solving for costs under given controls for general stochastic control problems with diffusion or jump diffusion type models. Discussions of the actual approximation method for various multiplexing-type systems is in [8, 5].

The idea is to approximate (2.1) with cost (2.3) or (2.4) by a suitable controlled finite state Markov chain on a state space which is a "discretization" of G. With suitable choices of the chain and associated cost function, the value of the associated optimal cost approximates that of the original problem arbitrarily well. (Similarly, if the control is fixed.) The chain is parameterized by a parameter h (analogous to a finite difference interval) such that as $h \to 0$, the "local" properties of the chain resemble more and more closely those of the diffusion. The state space G_h of the approximating chain is essentially the regular $h-$grid on G. (Actually, it is slightly larger, the total number of grid points being

$$\Pi_{i=1}^{d} \left[\left(X_i^+ - X_i^- \right]/h \right) + 3 \right],$$

with two of the extra points being associated with the "numerical reflecting boundary." The value of h is an input quantity, to be supplied by the user, as described below.

Solution techniques. The approximating Markov chain control problem is solved by the approximation in policy space method [3, 7]. This method

works by getting a sequence of control policies with decreasing cost, until the convergence criterion is satisfied. To insure against pathologies, the user specifies the maximum number of policy updates allowed. For each control policy, one needs to get an approximate solution to the linear equation for the cost (discounted cost problem) or "relative cost" (ergodic cost problem). This is the main computational work.

The user can specify one of several options for obtaining these approximate solutions. The simplest method is by use of a Gauss-Seidel relaxation. The code allows the use of overrelaxation, which generally (but not always) provides faster convergence. The user specifies the number of relaxations per policy update as well as the overrelaxation parameter. To use this direct relaxation, set the "number of multigrid sublevels" input parameter to zero. Good values of the overrelaxation parameter are problem dependent, and the user must do a little experimentation. The best values tend to be slightly less than the value at which the algorithm becomes unstable. For the problems in [5], we generally used values in the range $[1.1, 1.25]$. The range of good values tends to be similar for problems of similar structure, so good values for one member of a sequence of runs for closely related problems will generally be good for the other members. Assigning values of 1.0 to the overrelaxation parameters results in simple Gauss-Seidel iterations.

There is also the option of a form of the multigrid method, which is generally faster [1, 2, 7], and which we use whenever possible. The number of multigrid sublevels (i.e., grid coarsenings) must be specified, up to a user determined maximum value supplied at compile time. The present default value is three. See Section 3.3 for more information about changing the maximum number of multigrid sublevels. If "number of multigrid levels" equals zero, then only a Gauss-Seidel procedure (overrelaxed, if desired) will be used, with no multigrid sublevels. Within each level of multigrid, the code uses Gauss-Seidel or overrelaxed Gauss-Seidel smoothings. The user must specify the number of relaxations and the overrelaxation parameter for each multigrid level, as well as the maximum number of multigrid cycles allowed. The code does a policy update at the end of each full multigrid cycle. Thus, the maximum number of policy updates is defined by the maximum number of multigrid cycles allowed. See [4] for a general introduction to the multigrid method, and [7] for a brief summary. The method was introduced into the stochastic control area by Akian and Quadrat [1, 2]. Note that the use of multigrid increases the required memory, up to at most twice what is

required without it.

Program execution will stop when either the maximum number of control policy updates (or, equivalently, full multigrid cycles) has been reached or the (sup) norm of the difference between successive control updates has reached its desired value, whichever comes first.

State space description. The hyperrectangle G is described in the following way. We work in terms of an "origin" of G, and the discretization level h. This approach simplifies the code. The user enters the number of grid intervals $\{N_i^-, N_i^+\}$ for the negative and positive directions from the problem origin for each of the coordinate directions i, as well as the discretization level h, and the location X_0 of the origin of G. Thus $X_i^{\pm} = X_0 + hN_i^{\pm}$. One can always select the origin to be, say, the "lower left hand corner" of G, in which case $N_i^- = 0$, all i.

Some care must be exercised in choosing $X_0, N_i^{\pm}$, when there are multigrid sublevels. Each successive sublevel of the multigrid procedure uses a state space or grid whose spacing is twice that of the previous higher level. Thus, if m denotes the number of multigrid sublevels desired, the user should ascertain that the $[N_i^+ + N_i^-]/[2^k]$, $k = 1, \ldots, m$ for each dimension i are all integers. If more multigrid sublevels are specified than are computationally consistent, then the program will reduce the number of levels which are to be used such that consistency is maintained. Without multigrid usage, the state space may be described using an odd number of mesh intervals in any or all the dimensional discretizations for evaluation by overrelaxation.

Threshold and arbitrary controls. It is often desired to compute costs associated with a given control. The program can be used for this, since the user can input any desired control via an appropriately formatted data file. This control can be used in two ways. It can serve as the initial control of the policy updating routine. If a good initial controls were available, say from a previous run for related problem, this will save computational time. Alternatively, one can chose to have the entered control remain fixed throughout the computation, in which case the program will compute the values of the selected costs under that control. Input control is enabled using option 1 as described in subsection 3.6 and a brief description of the required format for the control input file is given in subsection 3.7.

In many applications, a threshold type of control is of interest due to the

simplicity of its implementation. To facilitate the computation of costs under a threshold control, one need not write a file containing all the control values. All that one needs to do is to select the threshold control option and put in the appropriate threshold parameters with the other data. A description of the threshold control and its parametrization is given under option 16 in subsection 3.6.

The values of the auxiliary or "relative cost function for the ergodic cost problem. With the Markov chain approximation method, the actual functions which are computed are costs for an approximating Markov chain problem (chosen by the program, using methods from [7]). In order to describe the quantities computed, it is useful to look briefly and formally at the Bellman equations for the control of a Markov chain with an ergodic cost function. Let $\bar{k}(x,u)$ be a cost realized by the chain when in state x and control $u(x)$ is used. For the ergodic cost problem, the Bellman equation is [3, 7, 9]

$$V(x) = \min_{\alpha} \left[\sum_{y} p(x,y|\alpha) V(y) + \bar{k}(x,\alpha) - \bar{\gamma} \right], \tag{2.5}$$

where α varies over the set of allowed control values, and x, y vary over the points in the state space. $\bar{\gamma}$ is the optimal average cost per unit time. The function $V(x)$ is the "relative cost." For a fixed control $u(\cdot)$, the equation for the cost and relative cost $W(x,u)$ is

$$W(x,u) = \left[\sum_{y} p(x,y|u(x)) W(y,u) + \bar{k}(x,u(x)) - \gamma(u) \right], \tag{2.6}$$

where $\gamma(u)$ is the average cost per unit time. Most of the computational work consists in solving a sequence of equations similar to (2.6). The solutions $V(\cdot)$ or $W(\cdot,u)$ are not unique, since if any solution is modified by adding a constant, then that new value will also solve (2.5) or (2.6), resp.

For the optimization problem, the program computes $\bar{\gamma}$, the numerical approximation to the optimal average cost for the original problem (2.1), (2.3). One selects a "centering point, called X_c, discussed in more detail below, and the program computes the relative cost $V(x) - V(X_c)$. A value of X_c must always be chosen for the ergodic cost problem. The values $V(x) - V(X_c)$ can be saved in a file (option 1024, where the output file is called *value.data*) as can the values of the computed controls (options 1 and 512),

as discussed in the program options Section 3.6. Analogous comments hold for the computation of $\gamma(u), W(\cdot, u)$ for the fixed control case.

The centering point X_c. For the ergodic cost problem, the matrix $P(u) = \{p(x, y|u(x)), x, y\}$ used in (2.5) and (2.6) is not a contraction and so the equation is solved slightly indirectly. The algorithm which is used takes advantage of the non-uniqueness of the solutions $V(x)$ and $W(x, u)$ in (2.5) and (2.6), resp., to transform the equations into "contractions" by the selection of a "centering point" X_c. See [7, Chapter 7] for a detailed description of the method. The selection of X_c is critical for good numerical behavior and some experimentation might be required. Good locations for one problem are typically good for similar problems. Poor choices lead to poor numerical behavior (slow convergence), and possibly even non-convergence. The value should be a point that is (loosely speaking) "as recurrent as possible" under what is expected to be the optimal control. For example, it should not be in a corner which is visited infrequently relative to other points. Often, the problem has an inherent stability. For example, see (3.1), where a good choice for $X_c = (x_{c,1}, x_{c,2})$ is $x_{c,2} = 0$ and $x_{c,1}$ being either zero or slightly positive. If there is a control or other dynamical force which pushes the path strongly out of a region, then X_c should not be located in that region.

The discounted cost problem (2.4) with reflecting boundaries. For the discounted cost problem, one may specify a centering point X_c or omit its usage. This is a run time option. When the centering point option is used, the numerical algorithm utilized is similar to the one used for the ergodic cost problem discussed above. Care must be exercised with the choice of X_c since a good selection for it will result in faster convergence. In addition to the use of X_c in the algorithm, the program will compute $V(X_c)$ and $V(x) - V(X_c)$, with the latter being saved in a file if desired. The output was chosen in this way since one often uses small discount factors which lead to large values of the costs, but moderate values for the differences between the costs at different points. It seemed preferable to center the values and print the large value for a single point (namely, X_c) only.

When the centering point option is **not** used (i.e., the centering point is not to be used in the numerical algorithm), the user must specify a sample point X_s at input time in lieu of the centering point. This is for "output" purposes only. The program outputs the value $V(X_s)$ and the user can also

save $V(x) - V(X_s)$ in a file (option 1024), if desired. The motivation for computing the differences of $V(x) - V(X_s)$ is the same as stated above.

2.2 The Absorbing Boundary Problem

For the absorbing boundary problem and a given continuous function $k(\cdot)$, the "cost rate" is defined in the form of (2.2). The cost function is then defined as

$$W(x,u) = E\int_0^\tau e^{-\beta t}\left[k(x(s),u(x(s)))ds\right] + Ee^{-\beta\tau}g(x_\tau) \qquad (2.7)$$

where $\tau = \min\{t : x(t) \notin G\}$ and initial condition $x(0) = x$.

3 Creating an Executable Program

3.1 Program availability

The source code is available on the World Wide Web from the Lefschetz Center for Dynamical Systems of Division of Applied Mathematics at Brown University's home page (http://www.dam.brown.edu/lcds.html). Viewers should select the "Software" link to access the software and documentation. The source code routines and example include files have been bundled together with the Unix "shar" command into a single Unix shell archive file for distribution. Clicking on the highlighted "stochastic control software" link will initiate a file transfer of the bundled source code to the user.

Once the file *scmodels.shr1* has been received by the user, the individual source files must be extracted from it. The command

sh scmodels.shr1

is the means by which the files are extracted from the archive file *scmodels.shr1*. This command will create a directory called "sc_code" into which the individual source code files will be placed. Several subdirectories will also be created which contain the necessary example include files needed to construct the model programs described in this document. Any or all of the example programs may be compiled using the identical *makefile* supplied within each of the example subdirectories. The *makefile* provided with

the source code has been configured for Sun workstations and may require some editing for use on other machines. Reading the sample include files described in this report is probably the best method to familiarize oneself with the style and method of utilizing the software. The sample include files can be copied and/or edited to construct an executable program for the the user-specific model. Comments, bugs, and suggestions should be directed to djarvis@dam.brown.edu.

3.2 How to Define the Model

We next describe the method of putting the actual model information into the general program, and then show how to run the program and select from among the many options. In order to provide maximum flexibility without an excessively complex code, the user must supply certain information in a fixed (but simple and reasonable) format. The specific features of the model, such as the drift $b(x,u)$ and the $k(x,u)$ part of the cost function, are supplied to the code through the use of Fortran "include" files which minimize coding work while allowing a reasonable flexibility of the general computational model. This user-supplied code should follow traditional Fortran standards, especially the fixed source code form requirement that statements must be within column positions 7 through 72. Although free format source code is accepted by many compilers, this model code uses a fixed form to increase portability. There are a total of thirteen "include" files for this software and three of these, *user_timing.h*, *user_timevar.h*, and *user_timeout.h* are used for optional program performance timing. The file *user_probs.h* is used for the control problem when the original process of interest is a controlled Markov chain and not (2.1). See Sections 7 and 8 for an explanation of the continuous Markov chain problem and its implementation. Problems having absorbing boundary conditions require the use of *user_boundary.h* to describe the boundary function $g(x)$. See Section 6 for details and an example. Otherwise, the remaining seven "include" files (listed first below) are used to completely describe a model of type (2.1) in software. Since most compilers will seek to incorporate the include files during the compilation process, all the include files should be present in the directory even if the files themselves are not being used (i.e., are empty) for the model.

Fortran "include" files

- user_drift.h: specifies the drift function $b(x, u)$
- user_cost.h: specifies cost $k(x, u)$
- user_covar.h: calculations of covariance Σ for model, if any
- user_var.h: user introduced variables in above files, if any
- user_in.h: input statements for user-introduced variables, if any
- user_out.h: output statements for user-introduced variables, if desired
- user_init.h: calculated constants or initializations for model, if any
- user_timing.h: machine dependent Fortran timing function, if used
- user_timevar.h: user introduced variable(s) for timing routine, if used
- user_timeout.h: output statement(s) for timing data, if used
- user_boundary.h: absorbing boundary function $g(x)$, if used
- user_kbdout.h: output statements for keyboard entered data, if desired
- user_probs.h: transition probabilities for the Markov chain model

There is a required notation for certain variables used in the "include" files because these files are incorporated directly into the code and the notation must be consistent. The list below describes these variables with a general outline of their basic usage. An explicit example then follows. Note that all but one of the provided variables are indexed with the user providing proper values for the maximum values of the indices in most cases. **In the use of the control and cost variables, the index i is used as a system index and it must appear, and it must not be modified by the user.** The index i represents the vector spatial index for a given point which the code varies over all the mesh points. The software uses vector indexing to represent the dimensional data in an explicit linear fashion in order to increase portability and performance.

Fortran Variables with Fixed Notation

- x(j), j=1, ..., dim: the components of the state vector x.
  The variables x(j) appear only on the right-hand side in the user's Fortran statement in explicit form as x(1), x(2), etc.

- u(i,j), j=1, ..., M: M = number of controls, $M \leq 4$.
  The j-th control is denoted by u(i,j). The variable u(i,j) appears only on the right-hand side of statements in the *user_drift.h* file with the index j used. Index i represents the vector spatial index and must appear and must not be modified by the user. The code will iterate over this general spatial index.

- k(l,m), (l=1,...,dim is the coordinate, m=1,...,M is the control component): control coefficient matrix.
  This is the matrix K in the definition of $b(x,u)$ in (2.1). Any input entry can be either positive or negative. The parameter k(l,m) appears only on the right-hand side of statements in the *user_drift.h* file. But all entries k(l,m) of the matrix K must be assigned at data input time, even if their values are zero.

- drift.
  In usage, this variable will be embedded within a dimensionally dependent if-statement which is indexed by the variable j. See the examples. This variable appears only on the left-hand side of statements in the *user_drift.h* file.

- cov(l,m), (l,m = 1,...,dim): symmetric covariance matrix Σ.
  User-supplied input values or Fortran statements to assign covariance values to the specified model. This file is needed only if the user wishes to represent the covariance in terms of other parameters. This file gives the formulas relating them. If the covariance input values are simply a set of numbers, then the file can be left empty and the values inserted with the other data at run time. See the example below. For uncorrelated covariance (Σ is diagonal) only the diagonal elements need to be supplied. In the case where the covariance is not diagonal, only the nonzero upper tridiagonal values are to be supplied. The program will directly assign the lower diagonal covariance matrix elements from the specified upper triangular elements to assure that the matrix is

symmetrical. The variables cov(l,m) appear only on the left-hand side of statements in the *user_covar.h* file and all indices must be specified.

- c(l), l=1, ..., 3*dim: cost coefficients.
  This is the set of $c_i, \bar{c}_i$ in (2.2). The variables c(l) appear only on the right-hand side of statements in *user_cost.h* file with all indices specified. By convention, we assume that the number of control cost coefficients **equals** the number of controls and that the control cost coefficients appear **first** in use.

- cost(i): defines the cost components for $k(x, u)$.
  Variable to which model dependent cost formula is assigned. Again, index i represents the vector spatial index and must appear but not be modified by the user. The code will iterate over this general spatial index. The variable cost(i) appears only on the left-hand side of statements in the *user_cost.h* file.

- value(i): defines the absorbing boundary cost at a calculated vector boundary index. This variable appears only on the left-hand side of statements in the *user_boundary.h* file.

- xmin(j), j=1, ..., dim: the minimum boundary value for dimension j, as evaluated by the software. This variable appears only on the right-hand side of statements in the *user_boundary.h* file, if needed.

- xmax(j), j=1, ..., dim: the maximum boundary value for dimension j, as evaluated by the software. This variable appears only on the right-hand side of statements in the *user_boundary.h* file, if needed.

For the continuous time Markov chain model described in Sections 7 and 8, the user will also have to define the controlled transition rates $r(x, y|u(x))$. To facilitate this feature, additional system-defined variables are made available and must be assigned appropriate values. Refer to Sections 7 and 8 for more information.

Example. In order to illustrate the use of the include files and the conventions for utilizing both system-provided and user-defined variables, consider

the following two dimensional problem, which originally arose as a heavy traffic limit to a multiplexer problem [8, 5].

$$\begin{aligned} dx_1(t) &= (\nu x_2(t) - a)\, dt - u_1(s)dt + dW_1(t) + dL_1(t) - dU_1(t) \\ dx_2(t) &= -(\lambda + \mu)x_2(t)\, dt + dW_2(t) + dL_2 - dU_2. \end{aligned} \tag{3.1}$$

The covariance matrix is diagonal with elements $\sigma(1,1) = 0$ and $\sigma(2,2) = 2\,\lambda\,\mu/(\lambda + \mu)$. For this model, the drift and covariance are written in terms of other parameters: μ, ν, λ, a. For computational purposes, a specific value of $\sigma(2,2)$ can either be entered directly as an input quantity or evaluated by the code from the other model parameters if desired. In this example, we will evaluate the covariance values via the "include" file to illustrate the appropriate coding procedure.

Define the ergodic cost for the problem as:

$$\gamma(u) = \lim_T \frac{1}{T} E\left[l_1 U_1(T) + \int_0^T c(1)u_1(s)ds + \int_0^T c(2)x_1(s)ds \right]. \tag{3.2}$$

The "include" and data files. The user can compile one of two versions of the program for entering data. In the "silent" version, no message prompts for data are produced by the program. There, the user prepares an input file which contains all of the required data in the order specified below. In the other version, called the "prompted" style, the program "asks" the user to enter the necessary system variable data in the prescribed order by producing data message prompts. The STDERR descriptor for the WRITE statements in the file *user_in.h* below is used for the "prompted" input version. It assures that the particular message prompts supplied by the user will be written to the monitor screen for entering the data for the variables of the problem as the program sequences through its input statements.

An additional feature of the "prompted" program version is its creation of a file named *input.prompted* with all the entered system data values. A complete file record of inputs will be created if the file *user_kbdout.h* includes the necessary WRITE statements using the KBD output descriptor to write the input data for user-introduced variables to the file. The file *input.prompted* could then used directly as an input file for a subsequent run of the program. Each execution run of the "prompted" version of the code

overwrites an existing *input.prompted* file so it is the user's responsibility to save any previous *input.prompted* files of interest.

For this example, we will assume that the "prompted" version of the code will be compiled by the user. The "include" files supplied by the user would be as written follows:

- user_drift.h:

```
if( j .eq. 1 ) then
  drift = nu*x(2) − a + k(1,1)*u(i,1)
else
  drift = − (lambda + mu) * x(2)
endif
```

  Comment: The j-th component of the spatial variable is always referred to as x(j). The variable j is used by the code to select the appropriate dimension for the drift evaluation and must appear. In this example, there is only one control and and it is denoted by u(i,1), as required. As noted above, the index i must appear as given above for proper indexing of the control.

- user_cost.h:

```
cost(i) = c(1)*u(i,1) + c(2)*x(1)
```

  Comment. Note that the parts of the cost dealing with the reflection terms L and U are not included here. Since their structure is fixed, their weights l_i, v_i are entered with the other parameters. Again note the required use of index i in the control variable u(i,1). Otherwise, all indices appearing in the cost evaluation have been specified by the user.

- user_covar.h:

```
cov(2,2) = 2.*lambda*mu / (lambda + mu)
```

  Comment: In this example, the only nonzero covariance value is given in terms of other parameters, and the code above specifies the functional dependence. If the covariances were simply a set of given real

numbers, then this file would be left empty, and the covariance values would be entered at run time together with the other problem data. Unless the user uses this include file to explicitly evaluate a covariance value, the input covariance values entered at run-time are used. If a particular covariance value is computed in this include file and is also entered as data at run time, the form used in this include file will be the one used by the program.

- user_var.h:

  real lambda, mu, nu, a
  common lambda, mu, nu, a

  Comment: In this file we list the "user-supplied" parameters which were introduced in the above files. Values for these variables can be entered with the other data at run time by means of READ statements or can be assigned by the user in the *user_init.h* file. If there are no such parameters, then this file will be empty. The program **does not** use automatic typing of variables so **all** introduced variables must be defined without exception. Compiler complaints about undefined variables will follow if this rule is ignored. The code is designed to use 64-bit floating point precision in its calculations for improved numerical accuracy. All real (i.e., floating point) variables for a given model should be typecast as simple Fortran REAL unless some specific need arises for another precision. For example, the variables used in Sun workstation timing functions require 32-bit precision. See the comments which accompany the *user_timevar.h* file usage for additional example information.

  By using the "precision neutral" REAL type specification, the provided *makefile* will supply the appropriate compiler option to specify that all REAL variables are to be 64-bits in precision. The distinct advantage of linking the default Fortran REAL variable typing with the 64-bit precision compiler option is that code portability is enhanced since no assumption is made with respect to the default number of bits used by REAL Fortran variables on a computational platform. This resolves the portability and maintenance problems that using DOUBLE PRECISION declarations pose since this results in 64 bits on many workstations but 128 bits on high-performance computers such as the

Cray C90.

- user_in.h:

```
write(STDERR,*) 'lambda mu nu a ?'
read(5,*) lambda, mu, nu, a
```

Comment: This file provides the necessary statements for reading data into the variables which the user has introduced for the model. The inclusion of the write statement is for the case where the user desires prompted keyboard messages for data input. See the next section for more on the use of STDERR with write statements. If no such prompting is desired, omit the write statement.

- user_out.h:

```
write(6,*) 'lambda = ', lambda
write(6,*) ' mu = ', mu
write(6,*) ' nu = ', nu
write(6,*) ' a = ', a
```

Comment: To document the executed problem, this file allows the user to provide output statements to print variable values of interest which have been supplied to describe the model. The manner and style in which this data is printed is left to the user.

- user_init.h:

Comment. This illustrative example does not require any variable initializations so the *user_init.h* file would be left empty. To demonstrate how this file might be used, suppose that we have a model with a cost defined by $k(x,u) =$ (b / N) $\sqrt{\text{Q}}$ $g(x,u)$, where b, Q, and N are input parameters that might vary from run to run and $g(x,u)$ is some function. We may reduce the execution time by writing $k(x,u) =$ M $g(x,u)$, and calculating M only once:

```
M = (b / N) * sqrt(Q)
```

where M, b, Q, and N have been declared as REAL Fortran variables in the file *user_vars.h*. Recall that REAL variables will be cast with 64-bit representation at compile time. Additionally, users may directly assign values to their model dependent variables in this file.

- user_timing.h:

  time = dtime(t)

  Comment. The "dtime" function is used on Sun workstations for timing purposes. The variables associated with the function as well as the function itself must defined for the code. For computational platforms other than Sun, check local documentation for the vendor-specific timing function call. See below for the proper declarations of the timing variables and the function itself.

- user_timevar.h:

  ```
  real*4 t(2), time, dtime
  common /time/ t, time
  ```

  Comment. The particular timing function and its variables are defined for the code. The Sun function "dtime " and its associated variables require 32-bit single precision real variables. To enforce this precision we use the REAL*4 declaration. The REAL*4 declaration informs the compiler that the defined variables and the function "dtime" are 4 bytes (i.e.. 32 bits) in precision. Note that the function name "dtime" is **omitted** from the Fortran COMMON statement since it is **not** a variable.

- user_timeout.h:

  ```
  write(6,*) 'Program time = ', time(1)+time(2), ' secs'
  write(6,*) ' user time = ', time(1), ' secs'
  write(6,*) ' system time = ', time(2), ' secs'
  ```

  Comment. Print the results of the timing calls for the program. In this example, the full output of the timing calls will be printed as specified

by the Sun function "dtime". Otherwise, the style and format of the output is left to the individual user.

- user_kbdout.h:

  write(KBD,*) lambda, mu, nu, a

  Comment: This file provides the necessary statements for writing user-supplied data to the output file which is always called *input.prompted* and which is created when the user compiles the code for prompted input. The data should be written in the same order as it is read into the program. Perhaps the simplest way to generate this include file is by changing all the READ statements in the *user_in.h* file to WRITE statements which use the KBD descriptor. Otherwise, the style and format of the output is left to the individual user. Note that the descriptor KBD is used to direct the output to the default file and must be used for all WRITE statements. If the program has been compiled without keyboard prompting then omit the WRITE statements, leaving this include file empty.

Since the parameters are input at run time, the proper sequence for entering the data will be deferred until we have discussed the compilation procedure..

3.3 Compiling and running the program

The UNIX software tool "make" is used to compile and generate the actual executable program. The "make" command is directed by the included *makefile* in compiling and linking the general source code routines into the specified program with as little user intervention as possible. Thus the user is relieved of both typing and remembering the compilation commands and options. The whole compilation and link process will be apparent to the user since "make" explicitly describes its progress as it executes. Table 1 below lists the appropriate commands to issue in order to construct the listed program.

The "Input mode" column in table 1 represents the user's preferred style of entering the problem data. A program compiled with the "Prompted"

Command	Program	Input mode	Dimension
make 2d	sctrl2d	Silent	Two
make 3d	sctrl3d	Silent	Three
make 4d	sctrl4d	Silent	Four
make 2dask	sctrl2dask	Prompted	Two
make 3dask	sctrl3dask	Prompted	Three
make 4dask	sctrl4dask	Prompted	Four

Table 1: Compilation choices for the control code

input mode will issue a sequence of prompts, at each of which the user will enter the appropriate data for defined system inputs. If users desire program prompts for their introduced variables then users must provide the necessary WRITE statments, as described in the previous section. The "Prompted" input mode will also save all system and introduced variables input data in a file named *input.prompted* for future reference. (This assumes that the user has supplied the necessary WRITE statements in the file *user_kbdout.h* to output the introduced variable data.) In the "Silent" input mode, the program quietly awaits input from either the keyboard or a file and no default input file is created. For either program version, output is directed (by default) to the terminal but may be saved by redirecting it to a file in Unix fashion, as illustrated below.

Note: Due to global variable dependencies and memory layout, it is necessary to utilize the "make clean" command if successive runs are for problems of different dimensions.

Example. As an example, consider the case where the user desires to experiment with a three dimensional model. First, let the input mode be "prompted." The user would type the command

make 3dask

to generate the prompted three-dimensional program. [The programs for two- and four-dimensions are created similarly.] Once the "make 3dask" command has been given, the source code will be compiled. The successful

compilation and linking creates the program named "sctrl3dask," which is the one to be executed by the user. Simply running

sctrl3dask

will cause the program to query the user for necessary parameter data, save those inputs in the default output file *input.prompted* and send the results to the terminal screen. [This file (perhaps modified by the user) can be used on a subsequent run as an input file, to save time.] Alternatively, by running

sctrl3dask > user.output

the user can save the program output in the file *user.output* while still inputting data at the program prompts. This command creates (or overwrites) the output file *user.output*. The output file contains the problem parameters, stopping data (number of cycles, etc.) and the numerical solutions. The user can also input data from a file and save the output by running

sctrl3dask < input.data > user2.output

thus directing data from the file *input.data* to the "sctrl3dask" program and saving (i.e., redirecting) the output from the terminal to the user-named file *user2.output*. This is the preferred method of entering data, especially when using the "silent" (i.e., unprompted) program versions, since it is the quickest and allows the user to verify input data before it is entered and used by a program. Issuing the command

make clean

command removes the previously created "include" files (those suffixed with a .h) and compiled object files (suffixed with an .o) thereby cleaning out these old files to avoid any possible global memory confusion, and saving disk space as well.

The software package has been extensively tested within its Sun Sparc10 workstation development environment but it is possible that compilation problems will appear when the code is ported to other machines. Such problems are usually the result of local system features such as the Fortran compiler name and library function names. System error messages can be very helpful in determining the nature of the difficulty. If the user is unable to

resolve a compilation problem, it is suggested that help be secured from local support personnel or more experienced Fortran users.

Important default settings. There are three important default settings at compilation time which the user should carefully review. The first setting to note is the maximum number of multigrid sublevels allowed. The present default value allows three multigrid sublevels (three levels of multigrid refinement). Experience has demonstrated that two or three multigrid levels tends to result in the shortest execution time for a program. This default value may be altered by changing the defined GRIDS value in the included *makefile* to the new maximum number of levels desired. Any number is possible, but the number is an upper bound to what can be entered as the "number of multigrid sublevels" (the actual numer to be used in the compiled program) in the data input.

To allow prompted input messages to appear for keyboard data entry when the program output is being redirected to a file, the program writes these messages to Fortran standard error output unit (the monitor). On the Sun Sparc 10, this is Fortran unit 0. Since this definition of the standard error output unit may be vendor dependent, this value may be altered by changing the defined ERR_OUTPUT parameter in the *makefile*. The value assigned to the ERR_OUTPUT parameter will be inherited by the STDERR output descriptor used by the WRITE statements for issuing of error and warning messages as well as interactive keyboard prompts, if used. See the vendor's Fortran documentation for the appropriate local standard error unit value. As a practical matter, if the prompted data input program is used and the messages don't appear on the monitor screen, then the value for ERR_OUTPUT is probably incorrect.

The third default setting is the parameter MAX_IBITS. This value represents the highest order bit value for machine integer representation and it is hardware dependent. By default, the parameter MAX_IBITS is set to 31 since the Sun workstation on which the software was developed and tested uses 32 bit integers. For most workstations this value of MAX_IBITS will be sufficient since the 32 bit integer representation is the one most commonly used. On a high performance machine such as the Cray C90, the value of MAX_IBITS would be set to 63 since the standard machine word length of a full integer is 64 bits.

3.4 Program inputs

As noted above, the input data can be entered either from the keyboard or from an existing file. The data is read by the program without format restrictions but it must be entered in the order specified below, and using the data types as documented below. Failure to use proper data types may result in erroneous results.

The stopping criterion tolerance represents the maximum absolute difference for the finest mesh between successive policy updates or between successive full multigrid cycles if the control is not updated. A fairly small value for this parameter should be chosen (e.g., $10^{-6} - 10^{-9}$).

Values for the "run time options" input parameter are described further in the next subsection. The following input description is for the reflecting boundary case, for either the ergodic or the discounted cost function. Recall that if the cost is of the discounted type, then there is the option of not using the centering point X_c in the numerical algorithm. Even if the option of not using the centering point in the computation is elected, then an input value of X_c is still needed, since the output is given in the form $V(X_c), V(x) - V(X_c)$.

General input data order and data types.
Reflecting Boundaries

- dimension of state variable (integer $=d$)
- Number of mesh intervals $(N_1^-, N_1^+), \ldots, (N_d^-, N_d^+))$. d lines.(integers)
- Number of multigrid sublevels (integer)
- Number of controls ($M \leq 4$) (integer)
- Non-diagonal covariance matrix? (integer: if 0 then NO, else YES)
- h, mesh interval width (real)
- Origin of the state space (real) X_0
- Value for run-time options (integer) [See Section 3.6 for full details]
- Centering point value X_c (real)
- Covariance values: upper triangular (real)
  d lines: $(\sigma(1,1), \ldots, \sigma(1,d))$, $(\sigma(2,2), \ldots, \sigma(2,d)), \ldots, (\sigma(d,d))$
  Note: This data line is still needed, even if the include file *user_covar.h* is used. Any calculations defined in *user_covar.h* will overwrite the corresponding values entered at input time.
- Drift control coefficient matrix K (real: d x (No. of controls) matrix)
  This entry has d lines containing $(k_{1,1}, \ldots, k_{1,M}), \ldots, (k_{d,1}, \ldots, k_{d,M})$.
- Maximum values of the controls $\bar{u}_1, \ldots, \bar{u}_M$ (real: no. of controls)

- Number of nonzero cost coefficients (number of nonzero $c_i, \bar{c}_i$ in (2.2), called $c(i)$ below) (integer)
- Cost coefficients values $c(1), \ldots,$ (real)
- User-supplied model inputs, if any [In the example, these are λ, μ, ν, a] The number of lines used is specified by the input statement
- Cost coefficients for underflow $l_1, \ldots, l_d$ (d real)
- Cost coefficients for overflow $v_1, \ldots, v_d$ (d real)
- Underflow boundary reflection values. (real: d times d) d lines, containing $p_1, \ldots, p_d$
- Overflow boundary reflection values. (real: d times d) d lines, containing $q_1, \ldots, q_d$
- Discount factor β (real; 0. = no discounting)
- Maximum number of policy updating steps (integer)
- Stopping criterion tolerance (real)
- Number of relaxations to be done for each multigrid level (integers)
- Overrelaxation factor for each multigrid level (reals)
- Threshold values (real, integer: no. of controls) These inputs are read only if the threshold option (option 16) is selected. See Section 3.6 for specifying this option. There are M lines, corresponding to the M controls. With the threshold control option, the $i-$th control takes the value $\bar{u}_i$ if some specified state component equals or exceeds a given real number. Otherwise the $i-$th control takes the value zero. The $i-$th line is (real, integer), the integer being the state component, and the real number is the threshold level. If some control is never to be active, set the threshold level large.

3.5 Parameter Inputs: Example 1 Returned

We now show how the parameters are put in, either via a file or via the monitor and keyboard. For any prompted program version, all data entered from the keyboard will be written to a default output file named *input.prompted* which is created by the program.

The first six inputs, which are dimension, number of mesh intervals, number of multigrid sublevels, number of controls, and the indicator that there is a non-diagonal covariance matrix allow the program to determine the amount of memory which is to be allocated for the problem. This tailoring of memory usage for a problem allows efficient memory management for larger memory problems as well as improving data locality for reducing execution time.

Given below is a sample input file for the two-dimensional problem (3.1), (3.2). Note that the comments to the right of the numerical values (beginning with an "!") have been added to clarify as well as demonstrate typical input data. In actual usage, no comments will appear in the input files. The run-time options value 2564 used in the example assures that all components of the cost will be calculated, the final controls with their spatial locations will be saved in a file, and the progress of the calculation will be sent to the output. The example in the next section (Section 3.6) shows how this particular value has been computed. Also, note that any component of the covariance which is entered both below and calculated in the *user_cov.h* include file will take the value calculated by the include file, since such calculations take place after input.

Sample input file for 2D model (3.1) and cost (3.2).

```
2                   ! program dimension
0 16                ! number of mesh intervals (N_1^-, N_1^+)
32 32               ! number of mesh intervals (N_2^-, N_2^+)
2                   ! number of multigrid sublevels
1                   ! number of controls
0                   ! non-diagonal covariance matrix (0 if false)
0.1154700538        ! mesh width h ≈ 1/√75
0. 0.               ! origin of the state space X_0 = (0., 0.)
2564                ! run-time options
0.1154700538 0.     ! location of centering point X_c = (h, 0.)
0.                  ! covariance value σ_11
0.                  ! covariance value σ_22
-1.                 ! control coefficient k(1,1) in drift eqn for x(1)
0.                  ! control coefficient k(2,1) in drift eqn for x(2)
0.4                 ! maximum control value ū_1
2                   ! number of cost coefficients c(j)
1. 0.               ! values for cost coefficients c(j)
0.2 1. 1. 0.48      ! λ μ ν a
0. 0.               ! underflow cost coefficients: {l_1, l_2}
200. 0.             ! overflow cost coefficients: {v_1, v_2}
1. 0.               ! p_1, dimension 1 underflow reflection
0. 1.               ! p_2, dimension 2 underflow reflection
-1. 0.              ! q_1, dimension 1 overflow reflection
0. -1.              ! q_2, dimension 2 overflow reflection
0.                  ! discounted cost factor
150                 ! maximum policy updating steps
0.00000001          ! stopping criterion tolerance
5 5 5               ! number of relaxations per multigrid level
1.2 1.2 1.2         ! overrelaxation parameter for each level
```

3.6 Program options

At its simplest, the program solves the optimal control problem for the specified data. This is option 0. But the program can be used for other purposes. These are indicated by the options parameter in the input data. The user selects the options from the following list, adds the values of the option indicator, and uses this in the options input line. All option values are based on powers of 2 so that there is no ambiguity in interpreting the sum of the values of the individual options. In all cases the user is required to specify an option value as part of the input data. So to solve the optimization problem only, set the run-time option to 0. Otherwise calculate the desired option value, as described below. See Table 2 for a summary of the run-time options.

By the "components of the costs," we mean the costs with the individual $U_i(\cdot), L_i(\cdot), u_i(\cdot), k_i(\cdot)$ used for all terms with nonzero cost coefficients $v_i, l_i, c_i, \bar{c}_i$, resp. These costs associated with the components are not weighted by the cost coefficients, i.e., the appropriate nonzero cost coefficients are set to unity.

Program options: Selected at run time.

- Option = 0 directs the program to solve the problem for the optimum system cost.
- Option = 1 directs the program to save the final computed controls as output files. These could subsequently be used as control input files for another run where a cost is to be evaluated with a fixed control and some specified cost function. The number of files created equals the number of controls. For each control u_j, the program creates an output file *controlJ.data* which contains the values of u_j at the grid points. See subsection 3.7 for a description of manner in which the grid points are traversed.
- Option = 2 directs the program to read input control data from an existing file or files. This option is used when we wish to either evaluate the costs for a given control or else start the policy updating method from a given control. The input file or files must be named *controlJ.data*, where $J = 1, \ldots, M \leq 4$. The input control will be used as the initial control in the policy updating scheme unless the "no

control updating" option 32 is also specified. This option is generally used for controls that were computed and saved at a previous run, in particular a run where option 1 was used.

- Option = 4 directs the program to explicitly compute all cost components (for which the weights are non-zero) for the problem once the optimal control problem has been solved. This is done with the control fixed at its value at the final iteration. This option is important for getting full use out of the data. Generally, the components of the optimal costs (with weights set to unity) are as important as the optimal cost itself. For example, see [5], where one would generally like to know what the actual mean buffer overflow, mean control loss and mean buffer waiting time are for the given optimal control or an *a-priori* given control.

- Option = 8 directs the program to evaluate **only** the cost components for the specified system with the control fixed. (It does not compute an optimal control.) Combine with option = 2 or option = 16 if nonzero control values are desired. Otherwise a "zero" control will be used. The control used will not be updated.

- Option = 16 directs the program to use a given threshold control when evaluating system cost. This option requires additional inputs for the threshold levels See the discussion under "program inputs." Data for this option is entered last of all in the input sequence. There is no control updating under this option. It is included simply because in many applications one wishes to compute performance under standard simple controls, and to compare the costs to that under the optimal. With other options, we allow arbitrary controls to be used, either because we wish to know the performance under them or because we wish to start the policy updates with some given "good" control. This option 16 facilitates this process when the control is of a simple threshold type. Under this option, the control is held fixed through the program execution (no policy updating).

- Option = 32 directs the program to do **no** control updating. If this option is used alone (or with option 4), then the program computes the cost for the zero control. It also computes each of the components of the

cost, with unity weights. When this option is used in conjunction with option = 2, the control control remains fixed as read from the input file throughout the program's execution. Note that when the control is not updated, the variable "maximum number of policy updates" is the maximum number of total multigrid cycles allowed, so it is still an important parameter.

- Option = 64 directs the program to **omit** the centering point in evaluating the cost and its components. This cannot be elected for the ergodic cost function. This option might be enabled for the reflecting boundary case with the discounted cost function. The absorbing boundary problem (option 128) does not use the centering point. A sample point location X_s is still required input for the problem for output purposes.

- Option = 128 is used when the cost function is for absorbing boundary Since there are no reflection directions for this model, the number of required inputs is reduced. No centering point X_c is used for this model by default. A sample point location X_s is still required as input for the problem since the output is given as $V(X_s)$. If option 1024 is also specified then $V(x) - V(X_s)$ will be output. See Section 6 for an example.

- Option = 256 is used when the initial model is defined as a controlled Markov chain, not as a reflected diffusion process. See Sections 7 and 8 for a more complete description of the specifics for this option.

- Option = 512 is used mainly when we wish to keep the control data for postprocessing, such as plotting. The option directs the program to create an output file of the final control data which includes the spatial coordinates, as well as the associated control values. The output file is named *control.data*. For each spatial coordinate, the corresponding control is listed. See subsection 3.7 for a more complete description of the file organization.

- Option = 1024 directs the program to create a output file of the final value (cost or relative cost) data which includes the corresponding spatial coordinates. The output file is named *value.data*. For each spatial

coordinate, the corresponding value is listed. See subsection 3.7 for a more complete description of the file organization.

- Option = 2048 informs the user of the program's progress through the solution process. One output line per policy step is produced showing the present policy update cycle number, the l_∞ norm and Euclidean l_2 norm of the differences between the current cycle and the last, the value of the system cost at the current iteration, and $\log_{10}$ of this cost.

Option	Function
0	calculate optimal cost and control
1	write final control data
2	read input control data
4	calculate optimal cost and cost components
8	calculate only cost components
16	use threshold control
32	no control updating
64	no centering point
128	absorbing boundary conditions problem
256	user-supplied probability updating code
512	write final control and spatial data
1024	write final value and spatial data
2048	report computation progress

Table 2: Program run-time options

Suppose that a user wants to compute the cost and its components for a given model, save the final computed optimal control for review and post-processing, and also survey the algorithm's progress as it solves a model. Enabling the program's run-time options for full system component evaluation, output spatial and final control values, and computation progress will accomplish these goals. (These options were exactly the ones specified for use in the input data for example 1.) For these options, the user adds the specified option values and assigns the parameter a value of 2564. That is,

$$\begin{aligned} \text{options} &= \text{calculate cost} + \text{compute cost components} \\ &\quad + \text{save final control data} + \text{iteration progress} \\ &= 0 + 4 + 512 + 2048 = 2564. \end{aligned}$$

is the parameter for the desired options. Thus, the program allows the user to tailor the execution of the program to further the desired research goals. The user, of course, should exercise discretion when selecting options. For example, if the "no control updating" and "save spatial and final control values" options are specified, then the program creates the output file *control.data* that contains only zero control values. In the situation where conflicting options are specified (i.e., options 2 and 16), the program selects the former option and ignores the latter. If input control files are specified for use (option 2) but do not exist, the program will issue a complaint and stop execution. Appropriate program option choices allow the user to wisely manage the program direction, data collection, and associated system resources.

3.7 Program Output

As the program is executed, output is directed to the terminal. To save this output information, the user need only redirect the output to a file, as illustrated in the example in Section 3.3. The program output consists of three basic parts: program type and options, input parameter values, and computed results. The program type and options informs the user of the dimensionality and mesh configuration of the problem as well the selected options used by the program, if any. All input parameters are printed by the program to allow verification of the input data by the user. Finally, the output prints the results of the calculations. If the computation progress option 2048 has been selected, the output will contain an active account of the progress of the multigrid method for solving the problem. One key item of information is whether or not the program terminated before achieving the desired precision. If the program seems to be unstable, decrease the overrelaxation factors. If it has not achieved the desired precision, but seems stable, increase the maximum number of allowed policy updates and/or the number of relaxations. User experience is the best guide when assigning these parameters.

The output files for final control data, spatial and control data, and spatial and value data (specified by program options 2, 512, and 1024, resp.) are written in a standard row/column format. For the output file options which include the spatial data, each line of the file contains the coordinates of a specified point in the space and the value of the cost or controls at that point. Under option 1024, the sequence of data for each line of such a file is the spatial position $x = (x_1, \ldots, x_d)$, and the the associated cost value at that point. Under option 512, the value of the M controls appears in lieu of the cost value.

The output is written beginning with the least-valued point in the space and the space is traversed in the following standard way. Start with $x = (x_1, \ldots, x_d)$, where each x_i is at its lowest value, Then increase x_1 until the right boundary is reached. Then increment x_2 by a unit, return x_1 to its lowest value and repeat, Continue until x_2 reaches its maximum value. The increment x_3 by a unit and repeat and so on until all points are reached.

When option 1 is selected to save the control data, no spatial data is written but the physical space is still traversed in the same manner. Control data computed by other programs for input to this program must use the same data output scheme if the assignment of the input control is to be done correctly.

Furthermore, all input control files must have the total number of mesh points used at the finest level for the problem as the **first** line of input. Thus, input control data files created by other programs must likewise include it. This value is used as a consistency check for the model grid.

4 Example: 2-D correlated noise model

We now give another illustrative example, where the Wiener processes are correlated. The system equations are

$$\begin{aligned} dx_1(t) &= (-\lambda\, x_1(t) + \mu\, x_2(t))\, dt + dW_1(t)\, dt \\ dx_2(t) &= (\lambda - \lambda_0)\, x_1(t)\, dt - (\mu + \mu_0 + \lambda_0)\, x_2(t))\, dt - u_1(s)\, dt + dW_2(t)\, dt \end{aligned} \tag{4.1}$$

with covariance matrix

$$\begin{bmatrix} \lambda\bar{x}_1 + \mu\bar{x}_2 & -(\lambda\bar{x}_1 + \mu\bar{x}_2) \\ -(\lambda\bar{x}_1 + \mu\bar{x}_2) & (\lambda - \lambda_0)\bar{x}_1 + (\mu + \mu_0 - \lambda_0)\bar{x}_2 + \lambda_0 \end{bmatrix} \tag{4.2}$$

where

$$\bar{x}_1 = \frac{\mu\,\lambda_0}{\lambda(\mu_0+\lambda_0)+\lambda_0\mu}.$$

and

$$\bar{x}_2 = \frac{\lambda\,\lambda_0}{\lambda(\mu_0+\lambda_0)+\lambda_0\mu}$$

The cost function is

$$\lim_T \frac{1}{T} E \int_0^T [c(1)u_1(s) + c(2)\max[0, x_2(s) - B]]\, ds. \tag{4.3}$$

The user supplied include files follow.

- user_drift.h:

```
if( j .eq. 1 ) then
  drift = -lambda*x(1) + mu*x(2)
else
  drift = (lambda-lambda0)*x(1) - (mu+mu0+lambda0)*x(2)
&         + k(2,1)*u(i,1)
endif
```

  Comment: In this example, the drift equations have one control which is denoted by u(i,1). Since there is no control variable in the first equation but only in the second equation, the input values for {k(1,1), k(2,1)} would be {0., -1.} in this model. The equations are dimensionally indexed by the variable j as required.

- user_cost.h:

```
cost(i) = c(1)*u(i,1) + c(2)*MAX(0., x(2)-B)
```

  Comment: This cost description utilizes the Fortran "MAX" function to extract the nonnegative component of x(2)-B. The reflection cost terms L and U have a fixed structure so their weights l_i, v_i are again entered as run time data.

- user_init.h:

```
x1b = (mu*lambda0) / (lambda*(mu0 + lambda0) + lambda0*mu)
x2b = (lambda*lambda0) / (lambda*(mu0 + lambda0) + lambda0*mu)
```

Comment: Evaluate x1b and x2b once for the duration of the program's execution. These values will be used in the *user_covar.h* file as part of the covariance calculations.

- user_covar.h:

```
cov(1,1) = lambda*x1b + mu*x2b
cov(1,2) = -(lambda*x1b + mu*x2b)
cov(2,2) = (lambda-lambda0)*x1b
&          + (mu + mu0 - lambda0)*x2b + lambda0
```

Comment: Compute the covariance terms from the specified user parameters. Note that the term cov(2,1) is not evaluated. The code will assign the value of cov(1,2) to the term cov(2,1) as it executes to assure that the covariance matrix is symmetrical. The lower diagonal covariance elements always receive their assigned values from the corresponding symmetrical upper triangular elements to assure symmetry. The character "&" which appears in the second covariance equation description is placed in column 6 for Fortran statement continuation, according to Fortran 77 convention.

- user_var.h:

```
real lambda, mu, lambda0, mu0, B, x1b, x2b
common lambda, mu, lambda0, mu0, B, x1b, x2b
```

Comment: The variables which we have introduced for this model are declared and typed.

- user_in.h:

```
write(STDERR,*) 'lambda mu lambda0 mu0 B ?'
read(5,*) lambda, mu, lambda0, mu0, B
```

- user_out.h:

```
write(6,*) 'lambda = ', lambda
write(6,*) 'lambda0 = ', lambda0
write(6,*) ' mu = ', mu
write(6,*) ' mu0 = ', mu0
write(6,*) ' B = ', B
write(6,*) ' x1b = ', x1b
write(6,*) ' x2b = ', x2b
```

  Comment: Files *user_in.h* and *user_out.h* are the means by which we enter data to the code and output it for our specific model variables.

- user_kbdout.h:

```
write(KBD,*) lambda, mu, lambda0, mu0, B
```

  Comment: This file provides the necessary statements for writing user-supplied data to the default output file *input.prompted* by utilizing the KBD descriptor for the WRITE statements.

Below an example input file for the two-dimensional model is given. Note that the comments to the right of the numerical values (beginning with an "!") have been added to clarify as well as demonstrate typical input data. In actual usage, no comments appear in input files.

Sample input file for 2D correlated noise model

```
2                    ! program dimension
63 95                ! number of mesh intervals (N_1^-, N_1^+)
63 127               ! number of mesh intervals (N_2^-, N_2^+)
1                    ! number of multigrid sublevels
1                    ! number of controls
1                    ! non-diagonal covariance (0 if false)
0.0316227766         ! mesh width h ≈ 1/√1000
0. 0.                ! origin of the state space X_0
516                  ! run-time options
0. 0.                ! location of centering point X_c
0. 0.                ! covariance values for σ_11 and σ_12
0.                   ! covariance value for σ_22
0.                   ! control coefficient k(1,1) for drift eqn 1
-1.                  ! control coefficient k(2,1) for drift eqn 2
1.                   ! maximum control value ū_1
2                    ! number of cost coefficients c(j)
1. 500.              ! values for cost coefficients c(j)
0.5 1. 0.1 0.2 2.    ! λ μ λ_0 μ_0 B
0. 0.                ! underflow cost coefficients: l_1, l_2
0. 0.                ! overflow cost coefficients: v_1, v_2
1. 0.                ! p_1, dimension 1 underflow reflection
0. 1.                ! p_2, dimension 2 underflow reflection
-1. 0.               ! q_1, dimension 1 overflow reflection
0. -1.               ! q_2, dimension 2 overflow reflection
0.                   ! discounted cost factor
1500                 ! maximum policy updating steps
0.00000001           ! stopping criterion tolerance
5 5                  ! number of relaxations per multigrid level
1.1 1.1              ! overrelaxation parameter for each level
```

5 Example: 3-D problem

Here is another example, which arose in a study of a multiplexer with two user classes under heavy traffic [5]. The systems equations are:

$$\begin{aligned} dx_1(t) &= (\nu_1 x_2(t) + \nu_2 x_3(t) - a)\,dt - (u_1(s) + u_2(s))\,dt \\ &\quad + dW_2(t) + dW_3(t) + dL(t) - dU(t) \\ dx_j(t) &= -(\lambda_{j-1} + \mu_{j-1})x_j(t)\,dt + dW_{1j-1}(t),\ j = 2,3. \end{aligned} \tag{5.1}$$

The $W_{11}(\cdot)$, $W_{12}(\cdot)$, $W_2(\cdot)$ and $W_3(\cdot)$ are mutually independent Wiener processes with variances

$$E[W_2(1)]^2 = \left(\frac{\nu_1 b_1 \lambda_1}{\lambda_1 + \mu_1} + \frac{\nu_2 b_2 \lambda_2}{\lambda_2 + \mu_2}\right), \quad E[W_3(1)]^2 = \sigma^2 E W_2(1)$$

$$E[W_{1j-1}(1)]^2 = \frac{2b_{j-1}\lambda_{j-1}\mu_{j-1}}{\lambda_{j-1} + \mu_{j-1}}, \quad j = 2,3.$$

We define $E[W_1(1)]^2 = E[W_2(1)]^2 + E[W_3(1)]^2$. For v_1 and $c(i)$ non-negative, we use the stationary cost:

$$E v_1 U(1) + E\left[\int_0^1 \sum_{i=1}^{2} c(i)u_i(s) + \int_0^1 c(3)x_1(s)ds\right].$$

The threshold control option will be used for this problem.

- user_drift.h:

```
if( j .eq. 1 ) then
  drift = nu(1)*x(2) + nu(2)*x(3) - a
&       + k(1,1)*u(i,1) + k(1,2)*u(i,2)
elseif( j .eq. 2 ) then
  drift = - (lambda(1) + mu(1)) * x(2)
else
  drift = - (lambda(2) + mu(2)) * x(3)
endif
```

Comment: Here the model uses two controls u(i,1) and u(i,2) in its drift equations. The necessary vector index i appears as required, as

well as the dimension index j. The first drift description statement is continued to the following line by the use of the "&" character which is placed in column 6.

- user_cost.h:

```
cost(i) = c(1)*u(i,1) + c(2)*u(i,2) + c(3)*x(1)
```

- user_covar.h:

```
      cov(1,1) = (s2 + 1.0) *
     &           ( nu(1)*b(1)*lambda(1) / (lambda(1) + mu(1))
     &           + nu(2)*b(2)*lambda(2) / (lambda(2) + mu(2)) )
      cov(2,2) = 2.*lambda(1)*mu(1)*b(1) / (lambda(1) + mu(1))
      cov(3,3) = 2.*lambda(2)*mu(2)*b(2) / (lambda(2) + mu(2))
```

Comment: The evaluations for the covariance terms which are used in this model. Recall that the program could read the numerical values directly from input if the user so desired, in which case the file *user_covar.h* would be left empty.

- user_var.h:

```
real lambda(2), mu(2), nu(2), b(2), a, s2
common lambda, mu, nu, b, a, s2
```

Comment: We define and type the variables which our Fortran description of the model utilizes.

- user_in.h:

```
write(STDERR,*) 'lambda(1) mu(1) nu(1) b(1) ?'
read(5,*) lambda(1), mu(1), nu(1), b(1)
write(STDERR,*) 'lambda(2) mu(2) nu(2) b(2) ?'
read(5,*) lambda(2), mu(2), nu(2), b(2)
write(STDERR,*) 'a ?'
read(5,*) a
write(STDERR,*) 'sigma**2 ?'
read(5,*) s2
```

- user_out.h:

```
write(6,*) 'lambda{1,2} = ', lambda(1), lambda(2)
write(6,*) ' mu{1,2} = ', mu(1), mu(2)
write(6,*) ' nu{1,2} = ', nu(1), nu(2)
write(6,*) ' b{1,2} = ', b(1), b(2)
write(6,*) 'Heavy traffic constant a = ', a
write(6,*) 'sigma**2 constant = ', s2
```

  Comment: Allow the input of values for the introduced variables and output their values for reference by the program.

- user_kbdout.h:

```
write(KBD,*) lambda(1), mu(1), nu(1), b(1)
write(KBD,*) lambda(2), mu(2), nu(2), b(2)
write(KBD,*) a
write(KBD,*) s2
```

  Comment: This file provides the necessary statements for writing user-supplied data to the default output file *input.prompted* by utilizing the KBD descriptor for the WRITE statements.

Below an example input file for this three-dimensional model is given. Note that the comments to the right of the numerical values (beginning with an "!") have been added to clarify as well as demonstrate typical input data. In actual usage, no such comments appear in input files. Note again that although the σ_{ii} are specified, they may be overwritten by the calculations given in the *user_cov.h* include file. The reflection directions for this model are normal to the surface faces.

Sample input file for 3D multiplexer model

```
3                 ! dimension of state
0 16              ! number of mesh intervals (N_1^-, N_1^+)
32 32             ! number of mesh intervals (N_2^-, N_2^+)
32 32             ! number of mesh intervals (N_3^-, N_3^+)
2                 ! number of multigrid sublevels
2                 ! number of controls
```

0	! off-diagonal covariance matrix (0 if false)
0.05	! mesh width $h = 1/\sqrt{20}$
0. 0. 0.	! origin of the state space X_0
564	! run-time options
0.05 0. 0.	! location of centering point X_c
0.	! covariance value σ_{11}
0.	! covariance value σ_{22}
0.	! covariance value σ_{33}
-1. -1.	! drift eqn 1 control coeffs k(1,1), k(1,2)
0. 0.	! drift eqn 2 control coeffs k(2,1), k(2,2)
0. 0.	! drift eqn 3 control coeffs k(3,1), k(3,2)
0.2 0.4	! maximum control values $\bar{u}_1$ and $\bar{u}_2$
3	! number of cost coefficients $c(j)$
2.5 5. 1.	! values for cost coefficients $c(j)$
0.4 2. 1. 0.5	! λ_1 μ_1 ν_1 b_1
0.4 1. 1. 0.5	! λ_2 μ_2 ν_2 b_2
0.48	! heavy traffic constant a
0.	! sigma**2 constant
0. 0. 0.	! underflow cost coefficients: l_1, l_2, l_3
10. 0. 0.	! overflow cost coefficients: v_1, v_2, v_3
1. 0. 0.	! p_1 dimension 1 underflow reflection
0. 1. 0.	! p_2 dimension 2 underflow reflection
0. 0. 1.	! p_3 dimension 3 underflow reflection
-1. 0. 0.	! q_1 dimension 1 overflow reflection
0. -1. 0.	! q_2 dimension 2 overflow reflection
0. 0. -1.	! q_3 dimension 3 overflow reflection
0.	! discounted cost factor
50	! maximum policy updating steps
0.00000001	! stopping criterion tolerance
3 5 5	! number of relaxations per multigrid level
1.2 1.2 1.2	! overrelaxation parameter for each level
1.4 2	! control 1 threshold: active if $x_2 \geq 1.4$
1.5 3	! control 2 threshold: active if $x_3 \geq 1.5$

6 Example: absorbing boundary problem

The following example of a two dimensional control problem with an absorbing boundary will illustrate how the program is used for this case. The system is

$$dx_1(t) = (\nu x_2(t) - a)\,dt - u_1(s)dt + dW_1(t),$$
$$dx_2(t) = -(\lambda + \mu)x_2(t)\,dt + dW_2(t).$$

The covariance matrix is diagonal with elements $\sigma(1,1) = 0$ and $\sigma(2,2) = 2\,\lambda\,\mu/(\lambda + \mu)$. The (undiscounted, $\beta = 0$) cost is

$$W(x,u) = E\left[\int_0^\tau c(1)u_1(s)ds + \int_0^\tau c(2)x_1(s)ds\right] + Eg(x_\tau),$$

where $\tau = \min\{t : x(t) \notin G$. The boundary cost $g(x)$ is constant on the set where $x_1 = X_1^+$, it equals $x_1^2 + x_2^2$ where $x_2 = X_2^+$ and it equals $|x_1 + x_2|$ elsewhere.

- user_drift.h:

```
if( j .eq. 1 ) then
  drift = nu*x(2) - a + k(1,1)*u(i,1)
else
  drift = - (lambda + mu) * x(2)
endif
```

  Comment: No changes for the drift equations from the earlier multiplexer example.

- user_cost.h:

```
cost(i) = c(1)*u(i,1) + c(2)*x(1)
```

  Comment. Note that the part of the cost dealing with the absorbing boundary described by $g(x)$ is not included here because its structure is fixed. Otherwise, the cost rate $k(x, u(x))$ is the same as we used in the 2D multiplexer example. The control variable includes the index i as required in all models.

- user_covar.h:

  cov(2,2) = 2.*lambda*mu / (lambda + mu)

  Comment: The evaluation of the nonzero covariance is the same as in the 2D multiplexer example.

- user_boundary.h

```
if( x(1) .eq. xmax(1) ) then
  value(i) = x1bcost
elseif( x(2) .eq. xmax(2) ) then
  value(i) = x(1)**2 + x(2)**2
else
  value(i) = ABS( x(1) + x(2) )
endif
```

  Comment: We assign the respective costs to the absorbing boundary as required by our function $g(x)$. The spatial values x_j of each boundary point are available for use as well as the maximum and minimum boundary values for each dimension j (xmax(j) and xmin(j), respectively). Note the required use of the vector index i with the value array. The values for the xmax and xmin arrays are strictly local to the absorbing boundary evaluation routine and cannot be utilized elsewhere.

- user_var.h:

```
real lambda, mu, nu, a, x1bcost
common lambda, mu, nu, a, x1bcost
```

  Comment: In addition to the variables used for the drift and covariance terms of the multiplexer model, the variable "x1bcost" is introduced for assigning a constant cost to the x_1 overflow boundary.

- user_in.h:

```
write(STDERR,*) 'lambda mu nu a ?'
```

```
read(5,*) lambda, mu, nu, a
write(STDERR,*) 'x1 overflow cost constant ?'
read(5,*) x1bcost
```

- user_out.h:

```
write(6,*) 'lambda = ', lambda
write(6,*) 'mu = ', mu
write(6,*) 'nu = ', nu
write(6,*) 'a = ', a
write(6,*) 'x1 overflow cost constant = ', x1bcost
```

Comment: The input and output statements for this model again allow the input and output of user-defined variables needed in describing the model.

- user_kbdout.h:

```
write(KBD,*) lambda, mu, nu, a
write(KBD,*) x1bcost
```

Comment: This file provides the necessary statements for writing user-supplied data to the default output file *input.prompted* by utilizing the KBD descriptor for the WRITE statements.

Given below is a sample input file for the two-dimensional absorbing boundary problem. The principle difference for inputs to this type of problem is that there are no overflow and underflow cost coefficients and no boundary reflections terms. In general, the input file will require two less input lines for the overflow and underflow costs and (2*dim) fewer input lines for the boundary reflections. Hence, this 2D absorbing example has deleted six lines associated with the boundaries which appeared in the 2D reflecting case. As before, comments have been added for descriptive purposes and do not appear in actual input file usage.

Sample input file for 2D absorbing boundary model

2	! program dimension
0 16	! number of mesh intervals (N_1^-, N_1^+)
32 32	! number of mesh intervals (N_2^-, N_2^+)
2	! number of multigrid sublevels
1	! number of controls
0	! non-diagonal covariance matrix (0 if false)
0.1154700538	! mesh width $h \approx 1/\sqrt{75}$
0. 0.	! origin of the state space $X_0 = (0., 0.)$
644	! run-time options
0.1154700538 0.	! location of sample point $X_s = (h, 0.)$
0.	! covariance value σ_{11}
0.	! covariance value σ_{22}
-1.	! control coefficient k(1,1) in drift eqn for $x(1)$
0.	! control coefficient k(2,1) in drift eqn for $x(2)$
0.4	! maximum control value $\bar{u}_1$
2	! number of cost coefficients
1. 3.	! values for cost coefficients
0.2 1. 1. 0.48	! λ μ ν a
200.	! absorbing boundary cost for x_1 overflow
0.	! discounted cost factor
150	! maximum policy updating steps
0.00000001	! stopping criterion tolerance
5 5 5	! number of relaxations per multigrid level
1.2 1.2 1.2	! overrelaxation parameter for each level

7 Continuous Time Markov Chain Control Problems

The previous discussion concerned control problems for diffusion type models such as (2.1). Since the basic computational method involves approximation by appropriate controlled Markov chains, it is natural that the codes can be effectively used on problems that are posed originally as control problems on Markov chain models. The basic details are the same for the reflecting and the absorbing boundary cases. The main discussion will be for the reflecting boundary case, and then the few required alterations for the absorbing boundary will be given, as for the diffusion model.

The structure of the state space is the same as discussed for the Markov chain approximations in Section 2. It is an $h-$grid G_h on a $d-$ dimensional hyperrectangle $G = \Pi_{i=1}^{d}[X_i^-, X_i^+]$, with each point communicating only to its immediate neighbors, to be further described below. The value of h is an input parameter. We assume the following boundary behavior, analogous to what was used for (2.1). If a point is on the boundary and tries to move out of G_h, then it is immediately reflected back with the mean reflection directions being described as they were in Section 2, by the vectors $p_i, q_i, i = 1, \ldots, d$. Indeed, this behavior fits many continuous time Markov chain models which arise in telecommunications applications.

For each point $x \in G_h$, the behavior is defined in terms of controlled *transition rates* $r(x, y|u(x))$, where y varies over the neighbors of x. The rates are linear in the controls and there can be up to four controls, exactly as in Section 2. Let e_i denote the unit vector in the $i-$th coordinate direction. Then, for $x \in G_h$, the rates for movement along the coordinate axes are written as

$$r(x, x + e_l h|u) = r_l^+(x) + \sum_{m=1}^{M} k_{lm}^+ u_m, \quad l = 1, \ldots, d, \tag{7.1}$$

$$r(x, x - e_l h|u) = r_l^-(x) + \sum_{m=1}^{M} k_{lm}^- u_m, \quad l = 1, \ldots, d, \tag{7.2}$$

with analogous formulas for the "off-coordinate axes" rates such as $r(x, x + e_l h + e_m h|u(x))$ which the user will define in the include file *user_probs.h*, as illustrated below. One significant difference between this type of model and the diffusion model is that the parameters $k_{lm}^{\pm}$ are **not** system-defined variables. Thus, the user must either use explicit constants for all the cost coefficients in the file user_cost.h, or else define an array into which will be put input data for these variables. As in Section 2, there are $\bar{u}_l$ such that $0 \leq u_l(x) \leq \bar{u}_l$.

The cost function. For the diffusion model case with a reflecting boundary, the cost (2.3) or (2.4) was specified as the sum of two components: a component due to the reflection and one due to a cost rate $k(x, u(x))$. For the continuous time Markov chain problem, the cost is still defined as the sum of two components. The component due to the cost rate $k(\cdot)$ is as in (2.3)

and (2.4). The cost associated with boundary reflections, is of course accrued only when the process attempts to leave the state space and must be returned by a reflection. As in the diffusion case, the user will input data for the overflow and underflow costs, as well the reflection directions. The reflecting boundary cost structure differs from what was used in the diffusion case in the following way: the inputs l_i, u_i are now the actual cost increments associated with a reflection step, according to the boundary on which the reflection occurs.

Transitions in "off-coordinate axes" directions. We now describe the format for the transition rates in 'off-coordinate axes directions" such as $r(x, x + e_l h + e_m h|u), l < m$. In all such expressions the arguments for each vector pair $\{e_l, e_m\}$ will be written in lexicographic order, i.e., $l < m$. The Markov chain approximation to the model (2.1) involved such transitions only if the covariance matrix were not diagonal [7]. Recall that one of the data entries told the program whether the covariance matrix was diagonal or not. Suppose that $d = 2$, and $\sigma_{12} > 0$. Then the transition probabilities for the Markov chain approximation for the diffusion model satisfy $p(x, x + e_1 h + e_2 h|u) > 0$, $p(x, x - e_1 h - e_2 h|u) > 0$, but for the transitions for the other directions, we have $p(x, x + e_1 h - e_2 h|u) = 0$, $p(x, x - e_1 h + e_2 h|u) = 0$, and conversely if $\sigma_{12} < 0$. For the current Markov chain model, we use a similar structure (so that the same code can be used). Thus, for $d = 2$, either the $\{r(x, x + e_1 h + e_2 h|u),\ r(x, x - e_1 h - e_2 h|u)\}$ can be nonzero or $\{r(x, x + e_1 h - e_2 h|u),\ r(x, x - e_1 h + e_2 h|u)\}$ can be nonzero. For general d, for each pair $l < m$, either $\{r(x, x + e_l h + e_m h|u),\ r(x, x - e_l h - e_m h|u)\}$ can be nonzero or $\{r(x, x + e_l h - e_m h|u),\ r(x, x - e_l h + e_m h|u)\}$ can be nonzero.

The choice of the directions ("northeast, southwest" or "northwest, southeast") is specified by using the covariance matrix used for the diffusion model. The covariance itself has no meaning for this continuous time Markov chain model. But the signs of the off–diagonal (the upper triangular) $\sigma_{l,m}$ will be used to indicate the directional choices, as follows. Thus, if there are off-coordinate transitions possible for the chain, then for the "nondiagonal covariance matrix" line in the input data, write 1. The signs of the symmetrical off-diagonal σ_{lm} determine the allowed directions. The input values for the off-diagonal $\sigma_{lm}, l < m$, can be any values, provided only that their signs are correct. For example, if the directions $\{e_l + e_m,\ -e_l - e_m\}$ are to be selected, then $\sigma_{lm} > 0$, and conversely if $\{e_l - e_m,\ -e_l + e_m\}$ are to be

selected then $\sigma_{lm} \leq 0$. The input values for the diagonal σ_{ll} are unimportant so they can be any non-negative number.

Defining the transition rates. The formulas for the transition rates are provided by the user in the file *user_probs.h*. The system variables rlft(i,j) and rrht(i,j) are used only when the model is a controlled Markov chain, as described in this section. Evaluate the transition rate of a point x to its neighbors $x - e_j h$ in the array rlft(i,j) and to $x + e_j h$ in the array rrht(i,j) as follows:

$$\text{rlft(i,j)} = r(x, x - e_j h|u), \text{ for j=}1, \ldots, d$$

and

$$\text{rrht(i,j)} = r(x, x + e_j h|u), \text{ for j=}1, \ldots, d.$$

Note that the index i is used by the system as before for indexing the vector spatial representation of the grid data. It must be included in the file names, in order for the program to interpret the file properly. The dimensional index j must be specified by the user. The formulas for evaluating the transition rates can be as simple or complex as the model requires.

The program uses the same format for the specification of the transition rates in the "diagonal" directions. Arrays rrht(i,j) and rlft(i,j) are used for these values, where j=d+1,..., d+D, where D is just the binomial coefficient "choosing 2 out of d." In detail, the index j has the following meaning:

- Dimension d=2:
  j = 3 for $\{e_1, e_2\}$, D=1,
- Dimension d=3:
  j = 4,5,6 for $\{e_1, e_2\}$, $\{e_1, e_3\}$, $\{e_2, e_3\}$, respectively, D=3,
- Dimension d=4:
  j = 5,6,7 for $\{e_1, e_2\}$, $\{e_1, e_3\}$, $\{e_1, e_4\}$, respectively, and
  j = 8,9,10 for $\{e_2, e_3\}$, $\{e_2, e_4\}$, $\{e_3, e_4\}$, respectively, D=6.

For $\sigma_{lm} > 0$, the off-coordinate axes transitions are assigned as

$$\text{rrht(i,j)} = r(x, x + e_l h + e_m h|u), \text{ for j=}d+1, \ldots, d+D$$

and

$$\text{rlft(i,j)} = r(x, x - e_l h - e_m h|u), \text{ for j=}d+1, \ldots, d+D$$

Otherwise for $\sigma_{lm} \leq 0$, the off-coordinate axes transitions are assigned as

$$\text{rrht(i,j)} = r(x, x + e_l h - e_m h|u), \text{ for j=}d+1, \ldots, d+D$$

and

$$\text{rlft(i,j)} = r(x, x - e_l h + e_m h|u), \text{ for j=}d+1, \ldots, d+D.$$

8 Example: continuous Markov chain

In this example we will use reflecting boundary conditions with a cost assigned to the x_2 overflow boundary. The transition rates are

$$\begin{aligned} r(x, x + e_2 h|u) &= \lambda_0 \left(NP - x_1 - x_2\right)(1 - u_1) \\ r(x, x - e_2 h|u) &= \mu_0 \, x_2 \\ r(x, x + e_1 - e_2 h|u)) &= \mu \, x_2 \\ r(x, x - e_1 + e_2 h|u) &= \lambda \, x_1 \end{aligned} \tag{8.1}$$

and the cost function is similiar to the correlated noise example but with the addition of the x_2 boundary overflow cost term:

$$Ev_2 U(1) + \lim_T \frac{1}{T} E \int_0^T \left[c(1)u_1(s) + c(2)\max[0, x_2(s) - B]\right] ds.$$

Recall that the state space of the Markov chain is an $h-$grid.

- user_drift.h:

- user_covar.h:

  Comment: These files are left empty since they are unused by the code for the continuous time Markov chain model. The files must be present for the compilation process to succeed.

- user_probs.h:

```
rrht(i,2) = lambda0 * (NP−x(1)−x(2)) * ( 1.0 − u(i,1) )
rlft(i,2) = mu0 * x(2)
rrht(i,3) = mu * x(2)
rlft(i,3) = lambda * x(1)
```

Comment: The transition rates for rrht(i,2) and rlft(i,2) are easily coded. Since there are no transitions along the first coordinate direction, we omit these zero-value assignments for efficiency. A covariance value $\sigma_{12} \leq 0.$ is needed since the off-coordinate axes transitions are in the southeast and northwest directions for rrht(i,3) and rlft(i,3), respectively.

- user_cost.h:

```
cost(i) = c(1)*u(i,1) + c(2)*max(0., x(2)−B)
```

- user_var.h:

```
real lambda, mu, lambda0, mu0, B
integer NP

common lambda, mu, lambda0, mu0, B, NP
```

- user_in.h:

```
write(STDERR,*) 'lambda mu lambda0 mu0 B NP ?'
read(5,*) lambda, mu, lambda0, mu0, B, NP
```

- user_out.h:

```
write(6,*) 'lambda = ', lambda
write(6,*) 'lambda0 = ', lambda0
write(6,*) ' mu = ', mu
write(6,*) ' mu0 = ', mu0
write(6,*) ' B = ', B
write(6,*) ' NP = ', NP
```

Comment: As in previous examples, we define the model variables and the necessary input and output statements so that our specific data will be read and subsequentially written for reference.

- user_kbdout.h:

  write(KBD,*) lambda, mu, lambda0, mu0, B, NP

  Comment: This file provides the necessary statements for writing user-supplied data to the default output file *input.prompted* by utilizing the KBD descriptor for the WRITE statements.

Again, it is important to note that the control coefficient matrix K with elements k(i,j) is not used for the continuous time Markov chain model as it was in the the diffusion models. Thus, input files for these models will not have data for the k(i,j) control coefficients.

Sample input file for 2D continuous time Markov chain model

2	! program dimension
0 50	! number of mesh intervals (N_1^-, N_1^+)
0 60	! number of mesh intervals (N_2^-, N_2^+)
1	! number of multigrid sublevels
1	! number of controls
1	! off-diagonal covariance terms (0 if false)
0.1	! mesh width $h = 1/\sqrt{100}$
0. 2.	! origin of the state space X_0
772	! run-time options
4. 7.	! location of centering point X_c
0. 0.	! covariance values for σ_{11} and σ_{12}
0.	! covariance value for σ_{22}
0.5	! maximum control value $\bar{u}_1$
2	! number of cost coefficients $c(j)$
15. 2.	! values for cost coefficients $c(j)$
.5 1. .1 .2 6.3 200	! λ μ λ_0 μ_0 B NP
0. 0.	! underflow cost coefficients: l_1, l_2
0. 1.	! overflow cost coefficients: v_1, v_2
1. 0.	! p_1, dimension 1 underflow reflection
0. 1.	! p_2, dimension 2 underflow reflection
-1. 0.	! q_1, dimension 1 overflow reflection
0. -1.	! q_2, dimension 2 overflow reflection

```
0.          ! discounted cost factor
50          ! maximum policy updating steps
.00000001   ! stopping criterion tolerance
5 5         ! number of relaxations per multigrid level
1.0 1.0     ! overrelaxation parameter for each level
```

Absorbing boundary. The absorbing boundary for a continuous Markov chain follows exactly the same format as the diffusion model case. One need not specify the reflection directions p_i and q_i and reflection costs l_i and v_i but the file for the boundary cost $g(x)$ must be provided as before.

References

[1] M. Akian. Resolution numerique d'equations d'Hamilton-Jacobi-Bellman au moyen d'algorithmes multigrilles et d'iterations sur les politiques. In *Eighth Conference on Analysis and Optimization of Systems*, Antibes, France, 1988. INRIA.

[2] M. Akian. *Methodes Multigrilles en Controle Stochastique.* PhD thesis, University of Paris, 1990.

[3] D.P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models.* Prentice Hall, Englewood Cliffs, NJ, 1987.

[4] W.L. Briggs. *A Multigrid Tutorial.* SIAM, Philadelphia, 1987.

[5] H. J. Kushner, J. Yang and, D. Jarvis. Controlled and optimally controlled multiplexing systems: A numerical exploration. *QUESTA*, 20:255–291, 1995.

[6] H.J. Kushner. Numerical methods for stochastic control problems in continuous time. *SIAM J. Control and Optimization*, 28:999–1048, 1990.

[7] H.J. Kushner and P. Dupuis. *Numerical Methods for Stochastic Control Problems in Continuous Time.* Springer, New York and Berlin, 1992.

[8] H.J. Kushner and L.F. Martins. Numerical methods for controlled and uncontrolled multiplexing and queueing systems. *QUESTA*, 16:241–285, 1994.

[9] M.L. Puterman. Markov decision processes. In D.P. Heyman and M.J. Sobel, editors, *Stochastic Models, Volume 2*, chapter 8. North Holland, Amsterdam, 1991.